sides 'No. of cylinders' (in decimal) :Interleave value: (in decimal) @FREE Syntax: Free [devname] Usage : Displays number of free sectors on a device @GFX Syntax: RUN GFX(<funct><args>) Usage : Graphics interface package for BASIC09 to do compatible VDG graphics commands @GFX2 Syntax: RUN GFX2([path]<funct><args>) Usage : Graphics interface package for BASIC09 to handle ax: none Usage : graphics/ windo e on-line help to p topics @IDEN rmation from OS s = use single lin xecution directory a device @INKE utine to input a abort to the pro k to a memory ntents of text files @LOAD Syntax: Load <pathname> [...] Usage : Loads modules into memory @MAKDIR Syntax: Makdir <pathname> Usage : Creates a new directory file @MDIR Syntax: Mdir [e] Usage : Displays the present memory module directory Opts : e = print extended module directory @MERGE Syntax: Merge <path> ndard output @MFREE Synt RAM memory @MODPATCH module in memory from warnings -c = compare modul name = link to module C o e V = verify module M = ma ontype [opt] Usage : Set mo monitor m = monochrome m sage : Creates and links an OS CS Syntax: Procs [e] Usage m Opts : e = display all proc : Prints the current data directory path @PXD Syntax: Pxd [Usage : Prints the current execution directory path @RENAME Syntax: Rename <filename> <new filename> Usage : Gives the file or directory a new name @RUNB Syntax: Runb <i-code module> Usage : BASIC09 run time package @SETIME Syntax: Setime [yy/mm TPR Syntax: s to num @ ler @TMODE Syntax: Tmode [pathname] [params] Usage : Displays or changes the operating parameters of the terminal @TUNEPORT Tuneport </t1 or /p> [value] Adjust the baud value for the serial port @UNLINK Syntax: Unlink <modname> Usage : Unlinks module(s) from memory @WCREATE Syntax:

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-1284
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
	Fax Messages	(07) 372-8325
SUPPORT	Brisbane OS9 Users Group	

Addresses for Correspondence

Editorial Material:

Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

Volume 5 December 1991 Number 11

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 5 Number 11

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

CHRISTMAS GREETINGS

Bob, Don, Jean-Pierre and myself would like to wish all members a merry Christmas and trust that you and your family enjoy a safe and happy new year. We would also like to thank you for your continued support of the OS9 Usergroup and we look forward to your active participation in the new year.

We know just how easy it is to leave the work to somebody else especially when we all seem to be governed by the higher priorities of life. I am sure that if every member made a commitment to submit just one small article during the coming year, our newsletter would be enjoyed far more by all. You don't have to be a technical guru to present something of interest to others. How about telling us what you use your computer for.

PUBLIC DOMAIN LIBRARY

Our PD library continues to grow at a steady rate thanks mainly to our overseas connections. Our OSK material has recently been enlarged by material from the European Usergroup. We know that the majority of our members run 6809 OS9 but for the OSK'ers here is some news.

The first 15 OSK-PD disks have now been entered into our PD-collection. It is the full TOP-collection collected by Wolfgang Ocker in Munich.

TOP stands for The Os9 Project and contains a lot of programmes ported from UNIX to OS9/68000. The TOP-disks will be of no use to 6809 users unless you'd be willing to (try to) port them to OS-9/6809 yourself. OSK users should note that some files might have to be recompiled. We do have some reports of trouble with some of the games which put too many linefeeds out. Also, the DU (Disk Usage) command assumes that all devices have 256-byte sectors, which was true for all OSK versions up to 2.3. The new version 2.4 allows any size sectors.

Anyhow, here's the list:

TOP #1 : Programs 1
TOP #2 : Programs 2
TOP #3 : Programs 3
TOP #4 : UUCP 1
TOP #5 : UUCP 2
TOP #6 : Documentation
TOP #7 : Sources 1
TOP #8 : Sources 2
TOP #9 : Sources 3
TOP #10 : Sources 4
TOP #11 : Sources 5
TOP #12 : Games 1
TOP #13 : Games 2
TOP #14 : Games 3
TOP #15 : Games 4

We'd recommend that you get all 15 disks, as the Sources, Docs and executable modules of any one programme are not necessarily on the same disk. We need to do a fair bit of sorting to achieve this, but that will take time. The disks are available only in Atari OSK 3.5" format at this stage.

IN THIS EDITION

We present a further chapter of the 'C' Tutorial series and have some further discussion notes downloaded from Bitnet by Don Berrie which we think may be of interest.

This December edition will have to keep you going until February next year as we will not produce one for January 1992, as has been the practice in previous years. We like to take a holiday too.

TIPS

We had a letter from a member in Sydney recently which posed a number of questions. We did get to reply even though it took a few weeks, sorry about that, my fault. Anyway, a couple of items may be of general interest.

Question:-
I know that the utility "error" will report the meaning of the error number, but I want this message to come up without having to type "error #xxx", can you help.
Answer:-
We are using a patched version of the Level 1 command "printerr" which works just fine. Printerr needs to be in your OS9Boot file and called from the "startup" script file. It looks for the text file "errmsg" in the /dd/SYS directory and reports the meaning of every error number (provided that the error number and text is in the "errmsg" file. The patch file is available in the PD library.

Question:-

Is there any way of telling if a VEF picture file is in squashed format?
Answer:-
The first character in a Standard VEF file is 00 while the first character in a squashed (compressed) file is 80. Info from M.V.Canvas version 2 manual.

Question:-
Is there any way of telling the size of a single file?
Answer:-
While the "dir -e" command will report the size (in hex) of every file in the current directory, we are not aware of a utility to report the size of an individual file. Perhaps somebody can help with this one. (Bob Devries wrote a utility named "fileinfo" which reports a lot of info about the file as well as checking the file integrity; perhaps we can persuade him to produce a simplified version for "filesize" ??

Until next year, Happy computing and Happy holiday.
Gordon.

oooooooooo0000000000ooooooooo

Using Kevin Darling's GFX2

by Bob Devries

Kevin Darling, the somewhat famous guru of the American OS9 community, has written a replacement GFX2 module for users of Basic09. The version I have, which also appears in our PD library, is some 122 bytes longer than the original module, but has quite a few extra commands. when IDENTed, this new GFX2 gives a CRC of $471CCE and a length of $949, whereas the old version has a CRC of $C940AD and a length of $8CA. It has only one failing, the FILL command is missing! If anyone has a later edition of this programme, please don't keep us in the dark!

I have written a little Basic09 programme to show you how to use the pulldown menus and mouse controls with kevin's GFX2. It is really simple to use, and does not require the use of complex variable types (structures). The programme opens a window (on /w as usual), which it then makes into a 40 x 24 graphics window using the DWSET call. After turning the cursor off, and selecting the window, the pulldown menu calls are used.

First, the main menu bar at the top of the window. This information is rarely seen, unless another window is opened on top of ours. Then the two titles for the pulldown menus are passed to the window manager, along with the numbers which they are to be identified by (33 and 34). Next come the items in the two menu pulldowns, with their identifiers, and whether they are enabled or not. You'll notice that the Format choice is carefully disabled, so you can't accidentally format something important! After all the information has been passed to the window manager (Windint), we issue the call 'WnSet' to tell it to draw the screen. Next, we set up the mouse handler with the 'SetMouse' call, and pass it values of 3,1,1 which are the scanrate, the timeout, and the autofollow values.

The programme next selects the pointer type using 'GCSet', and it chooses an arrow. Now we get to a loop which allows the selection of the pulldown. Notice, that the window manager does the actual pulldown for you and gives back to the programme the menu_id and menu_item of the selected item. On the basis of that, the programme goes to the appropriate subroutine and does what the pulldown suggests. Note, I have not included any error trapping, so if any of the commands (on the lines starting with 'Shell') are not either in memory or in the current execution directory, the programme will quit.

So you can see that it is really easy to make your own pulldown window system. Note that you do not need to run this from 'MultiVue', but you must have 'windint' in your OS9Boot file. If you don't, it will not work.

Regards, Bob Devries

```
PROCEDURE pulldown
 0000      DIM path:BYTE
 0007      DIM wd(3):STRING
 0013      DIM m1(3):STRING
 001F      DIM m2(2):STRING
 002B      DIM winstr:STRING[32]
 0037      DIM MId_Disk,MId_Mem:INTEGER
 0042      DIM Disable,Enable:INTEGER
 004D      DIM a:STRING[1]
 0059      DIM valid,fire,mx,my,area,sx,sy:INTEGER
 0078      DIM menu_id,menu_item:INTEGER
 0083      MId_Disk=33
 008A      MId_Mem=34
 0091      Disable=0
 0098      Enable=1
 009F      OPEN #path,"/w":UPDATE
 00AC      RUN getdevname(path,winstr)
 00BB      RUN gfx2(path,"DWSet",8,0,0,40,24,0,1,1)
 00E5      RUN gfx2(path,"select")
 00F8      RUN gfx2(path,"curoff")
 010B      RUN gfx2(path,"Title",wd,"Tools",34,10,2)
 0133      RUN gfx2(path,"Menu",wd,1,"Disk",MId_Disk,
8,4,m1,Enable)
 0168      RUN gfx2(path,"Item",m1,1,"Dir   ",Enable)
 018F      RUN gfx2(path,"Item",m1,2,"Free  ",Enable)
 01B6      RUN gfx2(path,"Item",m1,3,"PWD   ",Enable)
```

```
 01DD      RUN gfx2(path,"Item",m1,4,"Format",Disable)
 0204      RUN gfx2(path,"Menu",wd,2,"Memory",MId_Mem,
6,2,m2,Enable)
 023B      RUN gfx2(path,"Item",m2,1,"MFree",Enable)
 0261      RUN gfx2(path,"Item",m2,2,"Procs",Enable)
 0287      RUN gfx2(path,"WnSet",1,wd)
 02A1      RUN gfx2(path,"setmouse",3,1,1)
 02BF      RUN gfx2(path,"gcset",$CA,1)
 02D8      LOOP
 02DA        RUN gfx2(path,"onmouse",0)
 02F1        RUN gfx2(path,"mouse",valid,fire,mx,my,
area,sx,sy)
 0326        IF valid<>0 AND fire=1 AND area=1 THEN
 0340          RUN gfx2(path,"getsel",menu_id,menu_item)
 035D          IF menu_id=2 THEN
 0369            GOTO 100
 036B          ENDIF
 036F          IF menu_id=33 THEN
 037B            GOSUB 200
 037F          ENDIF
 0381          IF menu_id=34 THEN
 038D            GOSUB 300
 0391          ENDIF
 0393        ENDIF
 0395      ENDLOOP
 0399 100  RUN gfx2(1,"select")
 03AD      CLOSE #path
 03B3      END
 03B5 200  ON menu_item GOSUB 400,500,600,700
 03CF      RETURN
 03D1 300  ON menu_item GOSUB 800,900
 03E3      RETURN
 03E5 400  PRINT #path,CHR$(12)
 03F2      SHELL "dir"+")"+winstr
 0401      GOSUB 1000
 0405      RETURN
 0407 500  PRINT #path,CHR$(12)
 0414      SHELL "free"+")"+winstr
 0424      GOSUB 1000
 0428      RETURN
 042A 600  PRINT #path,CHR$(12)
 0437      SHELL "pwd"+")"+winstr
 0446      GOSUB 1000
 044A      RETURN
 044C 700  PRINT #path,CHR$(12)
 0459      RETURN
 045B 800  PRINT #path,CHR$(12)
 0468      SHELL "mfree"+")"+winstr
 0479      GOSUB 1000
 047D      RETURN
 047F 900  PRINT #path,CHR$(12)
 048C      SHELL "procs"+")"+winstr
 049D      GOSUB 1000
 04A1      RETURN
 04A3 1000 WHILE a="" DO
 04B2        RUN inkey(path,a)
 04C1      ENDWHILE
 04C5      RETURN

PROCEDURE getdevname
 0000      TYPE registers=cc,a,b,dp:BYTE; x,y,u:INTEGER
 0025      DIM regs:registers
 002E      PARAM wpath:BYTE
 0035      PARAM winnam:STRING[32]
 0041      DIM i:INTEGER
 0048      DIM callcode:BYTE
 004F      regs.a=wpath
 005B      regs.b=$0E
 0067      regs.x=ADDR(winnam)
 0075      callcode=$8D
 007D      RUN syscall(callcode,regs)
 008C      FOR i=1 TO 32
 009C      EXITIF MID$(winnam,i,1)>CHR$(128) THEN
 00AF        winnam="/"+LEFT$(winnam,i-
1)+CHR$(ASC(MID$(winnam,i,1))-128)
 00D1      ENDEXIT
 00D5      NEXT i
 00E0      END
```

ooooooooooOOOOOOOOOOoooooooooo

A C Tutorial
Chapter 5 - Functions and variables

OUR FIRST USER DEFINED FUNCTION

Load and examine the file SUMSQRES.C for an example of a C program with functions. Actually this is not the first function we have encountered because the "main" program we have been using all along is technically a function, as is the "printf" function. The "printf" function is a library function that was supplied with your compiler. Notice the executable part of this program. It begins with a line that simply says "header();", which is the way to call any function. The parentheses are required because the C compiler uses them to determine that it is a function call and not simply a misplaced variable.

When the program comes to this line of code, the function named "header" is called, its statements are executed, and control returns to the statement following this call. Continuing on we come to a "for" loop which will be executed 7 times and which calls another function named "square" each time through the loop, and finally a function named "ending" will be called and executed. For the moment ignore the "index" in the parentheses of the call to "square". We have seen that this program therefore calls a header, 7 square calls, and an ending.

Now we need to define the functions.

DEFINING THE FUNCTIONS

Following the main program you will see another program that follows all of the rules set forth so far for a "main" program except that it is named "header()". This is the function which is called from within the main program. Each of these statements are executed, and when they are all complete, control returns to the main program. The first statement sets the variable "sum" equal to zero because we will use it to accumulate a sum of squares. Since the variable "sum" is defined as an integer type variable prior to the main program, it is available to be used in any of the following functions. It is called a "global" variable, and it's scope is the entire program and all functions.

More will be said about the scope of variables at the end of this chapter. The next statement outputs a header message to the monitor. Program control then returns to the main program since there are no additional statements to execute in this function. It should be clear to you that the two executable lines from this function could be moved to the main program, replacing the header call, and the program would do exactly the same thing that it does as it is now written. This does not minimize the value of functions, it merely illustrates the operation of this simple function in a simple way. You will find functions to be very valuable in C programming.

PASSING A VALUE TO A FUNCTION

Going back to the main program, and the "for" loop specifically, we find the new construct from the end of the last lesson used in the last part of the for loop, namely the "index++". You should get used to seeing this, as you will see it a lot in C programs. In the call to the function "square", we have an added feature, namely the variable "index" within the parentheses. This is an indication to the compiler that when you go to the function, you wish to take along the value of index to use in the execution of that function. Looking ahead at the function "square", we find that another variable name is enclosed in its parentheses, namely the variable "number". This is the name we prefer to call the variable passed to the function when we are in the function. We can call it anything we wish as long as it follows the rules of naming an identifier.

Since the function must know what type the variable is, it is defined following the function name but before the opening brace of the function itself. Thus, the line containing "int number;" tells the function that the value passed to it will be an integer type variable. With all of that out of the way, we now have the value of index from the main program passed to the function "square", but renamed "number", and available for use within the function. Following the opening brace of the function, we define another variable "numsq" for use only within the function itself, (more about that later) and proceed with the required calculations. We set "numsq" equal to the square of number, then add numsq to the current total stored in "sum". Remember that "sum += numsq" is the same as "sum = sum + numsq" from the last lesson. We print the number and its square, and return to the main program.

MORE ABOUT PASSING A VALUE TO A FUNCTION

When we passed the value of "index" to the function, a little more happened than meets the eye. We did not actually pass the value of index to the function, we actually passed a copy of the value. In this way the original value is protected from accidental corruption by a called function. We could have modified the variable "number" in any way we wished in the function "square", and when we returned to the main program, "index" would not have been modified. We thus protect the value of a variable in the main program from being accidentally corrupted, but we cannot return a value to the main program from a function using this technique. We will find a well defined method of returning values to the main program or to any calling function when we get to arrays and another method when we get to pointers.

Until then the only way you will be able to communicate back to the calling function will be with global variables. We have already hinted at global variables above, and will discuss them in detail later in this chapter. Continuing in the main program, we come to the last function call, the call to "ending". This call simply calls the last function which has no local variables defined. It prints out a message with the value of "sum" contained in it to end the program. The program ends by returning to the main program and finding nothing else to do. Compile and run this program and observe the output.

NOW TO CONFESS A LITTLE LIE

I told you a short time ago that the only way to get a value back to the main program was through use of a global variable, but there is another way which we will discuss after you load and display the file named SQUARES.C. In this file we will see that it is simple to return a single value from a called function to the calling function. But once again, it is true that to return more than one value, we will need to study either arrays or pointers. In the main program, we define two integers and begin a "for" loop which will be executed 8 times. The first statement of the for loop is "y = squ(x);", which is a new and rather strange looking construct. From past experience, we should have no trouble understanding that the "squ(x)" portion of the statement is a call to the "squ" function taking along the value of "x" as a variable. Looking ahead to the

function itself we find that the function prefers to call the variable "in" and it proceeds to square the value of "in" and call the result "square".

Finally, a new kind of a statement appears, the "return" statement. The value within the parentheses is assigned to the function itself and is returned as a usable value in the main program. Thus, the function call "squ(x)" is assigned the value of the square and returned to the main program such that "y" is then set equal to that value. If "x" were therefore assigned the value 4 prior to this call, "y" would then be set to 16 as a result of this line of code. Another way to think of this is to consider the grouping of characters "squ(x)" as another variable with a value that is the square of "x", and this new variable can variables be used any place it is legal to use a variable of its type. The values of "x" and "y" are then printed out.

To illustrate that the grouping of "squ(x)" can be thought of as just another variable, another "for" loop is introduced in which the function call is placed in the print statement rather than assigning it to a new variable. One last point must be made, the type of variable returned must be defined in order to make sense of the data, but the compiler will default the type to integer if none is specified. If any other type is desired, it must be explicitly defined. How to do this will be demonstrated in the next example program. Compile and run this program.

FLOATING POINT FUNCTIONS

Load the program FLOATSQ.C for an example of a function with a floating point type of return. It begins by defining a global floating point variable we will use later. Then in the "main" part of the program, an integer is defined, followed by two floating point variables, and then by two strange looking definitions. The expressions "sqr()" and "glsqr()" look like function calls and they are. This is the proper way in C to define that a function will return a value that is not of the type "int", but of some other type, in this case "float". This tells the compiler that when a value is returned from either of these two functions, it will be of type "float". Now refer to the function "sqr" near the center of the listing and you will see that the function name is preceded by the name "float". This is an indication to the compiler that this function will return a value of type "float" to any program that calls it.

The function is now compatible with the call to it. The line following the function name contains "float inval;", which indicates to the compiler that the variable passed to this function from the calling program will be of type "float". The next function, namely "glsqr", will also return a "float" type variable, but it uses a global variable for input. It also does the squaring right within the return statement and therefore has no need to define a separate variable to store the product. The overall structure of this program should pose no problem and will not be discussed in any further detail. As is customary with all example programs, compile and run this program. There will be times that you will have a need for a function to return a pointer as a result of some calculation. There is a way to define a function so that it does just that. We haven't studied pointers yet, but we will soon. This is just a short preview of things to come.

SCOPE OF VARIABLES

Load the next program, SCOPE.C, and display it for a discussion of the scope of variables in a program. The first variable defined is a global variable "count" which is available to any function in the program since it is defined before any of the functions. In addition, it is always available because it does not come and go as the program is executed. (That will make sense shortly.) Farther down in the program, another global variable named "counter" is defined which is also global but is not available to the main program since it is defined following the main program. A global variable is any variable that is defined outside of any function. Note that both of these variables are sometimes referred to as external variables because they are external to any functions.

Return to the main program and you will see the variable "index" defined as an integer. Ignore the word "register" for the moment. This variable is only available within the main program because that is where it is defined. In addition, it is an "automatic" variable, which means that it only comes into existence when the function in which it is contained is invoked, and ceases to exist when the function is finished. This really means nothing here because the main program is always in operation, even when it gives control to another function. Another integer is defined within the "for" braces, namely "stuff". Any pairing of braces can contain a variable definition which will be valid and available only while the program is executing statements within those braces. The variable will be an "automatic" variable and will cease to exist when execution leaves the braces. This is convenient to use for a loop counter or some other very localized variable.

MORE ON "AUTOMATIC" VARIABLES

Observe the function named "head1". It contains a variable named "index", which has nothing to do with the "index" of the main program, except that both are automatic variables. When the program is not actually executing statements in this function, this variable named "index" does not even exist. When "head1" is called, the variable is generated, and when "head1" completes its task, the variable "index" is eliminated

completely from existence. Keep in mind however that this does not affect the variable of the same name in the main program, since it is a completely separate entity. Automatic variables therefore, are automatically generated and disposed of when needed. The important thing to remember is that from one call to a function to the next call, the value of an automatic variable is not preserved and must therefore be reinitialized.

WHAT ARE STATIC VARIABLES?

An additional variable type must be mentioned at this point, the "static" variable. By putting the reserved word "static" in front of a variable declaration within a function, the variable or variables in that declaration are static variables and will stay in existence from call to call of the particular function. By putting the same reserved word in front of an external variable, one outside of any function, it makes the variable private and not accessible to use in any other file. This implies that it is possible to refer to external variables in other separately compiled files, and that is true. Examples of this usage will be given in chapter 14 of this tutorial.

USING THE SAME NAME AGAIN

Refer to the function named "head2". It contains another definition of the variable named "count". Even though "count" has already been defined as a global variable, it is perfectly all right to reuse the name in this function. It is a completely new variable that has nothing to do with the global variable of the same name, and causes the global variable to be unavailable in this function. This allows you to write programs using existing functions without worrying about what names were used for variables in the functions because there can be no conflict. You only need to worry about the variables that interface with the functions.

WHAT IS A REGISTER VARIABLE?

Now to fulfill a promise made earlier about what a register variable is. A computer can keep data in a register or in memory. A register is much faster in operation than memory but there are very few registers available for the programmer to use. If there are certain variables that are used extensively in a program, you can designate that those variables are to be stored in a register if possible in order to speed up the execution of the program. Depending on the computer and the compiler, a small number of register variables may be allowed and are designated by putting the word "register" in front of the desired variable.

Check your compiler documentation for the availability of this feature and the number of register variables. Most compilers that do not have any register variables available, will simply ignore the word "register" and run normally, keeping all variables in memory. Register variables are only available for use with integer and character type variables. This may or may not include some of the other integer-like variables such as unsigned, long, or short. Check the documentation for your compiler.

WHERE DO I DEFINE VARIABLES?

Now for a refinement on a general rule stated earlier. When you have variables brought to a function as arguments to the function, they are defined immediately after the function name and prior to the opening brace for the program. Other variables used in the function are defined at the beginning of the function, immediately following the opening brace of the function, and before any executable statements.

STANDARD FUNCTION LIBRARIES

Every compiler comes with some standard predefined functions which are available for your use. These are mostly input/output functions, character and string manipulation functions, and math functions. We will cover most of these in subsequent chapters. In addition, most compilers have additional functions predefined that are not standard but allow the programmer to get the most out of his particular computer. In the case of the IBM-PC and compatibles, most of these functions allow the programmer to use the BIOS services available in the operating system, or to write directly to the video monitor or to any place in memory. These will not be covered in any detail as you will be able to study the unique aspects of your compiler on your own. Many of these kinds of functions are used in the example programs in chapter 14.

WHAT IS RECURSION?

Recursion is another of those programming techniques that seem very intimidating the first time you come across it, but if you will load and display the example program named RECURSON.C, we will take all of the mystery out of it. This is probably the simplest recursive program that it is possible to write and it is therefore a stupid program in actual practice, but for purposes of illustration, it is excellent. Recursion is nothing more than a function that calls itself. It is therefore in a loop which must have a way of terminating. In the program on your monitor, the variable "index" is set to 8, and is used as the argument to the function "count dn". The function simply decrements the variable, prints it out in a message, and if the variable is not zero, it calls itself, where it decrements it again, prints it, etc. etc. etc.

Finally, the variable will reach zero, and the

function will not call itself again. Instead, it will return to the prior time it called itself, and return again, until finally it will return to the main program and will return to DOS. For purposes of understanding you can think of it as having 8 copies of the function "count_dn" available and it simply called all of them one at a time, keeping track of which copy it was in at any given time. That is not what actually happened, but it is a reasonable illustration for you to begin understanding what it was really doing.

WHAT DID IT DO?

A better explanation of what actually happened is in order. When you called the function from itself, it stored all of the variables and all of the internal flags it needs to complete the function in a block somewhere. The next time it called itself, it did the same thing, creating and storing another block of everything it needed to complete that function call. It continued making these blocks and storing them away until it reached the last function when it started retrieving the blocks of data, and using them to complete each function call. The blocks were stored on an internal part of the computer called the "stack". This is a part of memory carefully organized to store data just as described above.

It is beyond the scope of this tutorial to describe the stack in detail, but it would be good for your programming experience to read some material describing the stack. A stack is used in nearly all modern computers for internal housekeeping chores. In using recursion, you may desire to write a program with indirect recursion as opposed to the direct recursion described above. Indirect recursion would be when a function "A" calls the function "B", which in turn calls "A", etc. This is entirely permissible, the system will take care of putting the necessary things on the stack and retrieving them when needed again. There is no reason why you could not have three functions calling each other in a circle, or four, or five, etc. The C compiler will take care of all of the details for you.

The thing you must remember about recursion is that at some point, something must go to zero, or reach some predefined point to terminate the loop. If not, you will have an infinite loop, and the stack will fill up and overflow, giving you an error and stopping the program rather abruptly.

ANOTHER EXAMPLE OF RECURSION

The program named BACKWARD.C is another example of recursion, so load it and display it on your screen. This program is similar to the last one except that it uses a character array. Each successive call to the function named "forward_and_backward" causes one character of the message to be printed. Additionally, each time the function ends, one of the characters is printed again, this time backwards as the string of recursive function calls is retraced. Don't worry about the character array defined in line 3 or the other new material presented here.

After you complete chapter 7 of this tutorial, this program will make sense. It was felt that introducing a second example of recursion was important so this file is included here. One additional feature is built into this program. If you observe the two calls to the function, and the function itself, you will see that the function name is spelled three different ways in the last few characters. The compiler doesn't care how they are spelled because it only uses the first 8 characters of the function name so as far as it is concerned, the function is named "forward_". The remaining characters are simply ignored. If your compiler uses more that 8 characters as being significant, you will need to change two of the names so that all three names are identical. Compile and run this program and observe the results.

PROGRAMMING EXERCISES

1. Rewrite TEMPCONV.C, from an earlier chapter, and move the temperature calculation to a function.

2. Write a program that writes your name on the monitor 10 times by calling a function to do the writing. Move the called function ahead of the "main" function to see if your compiler will allow it.

ooooooooooOOOOOOOOOOoooooooooo

An Index of Rainbow OS9 Articles
compiled by Bob Devries
January - December '87

January 1987 page 160
Get Comfortable with OS9 - Tutorial. A good introduction for beginners.
Nancy Ewart

January 1987 page 193
KISSable OS9 - Debunking some OS9 myths.
Dale L. Puckett

February 1987 page 26
Murder at the Hotel CoCo - Game A Rainbow staff imposter is bent on mayhem!

Dale Lear

February 1987 page 190
KISSable OS9 - A level II report.
Dale L. Puckett

February 1987 page 204
Pipes and Filters - Tutorial. The misunderstood features.
Bruce N. Warner

March 1987 page 186
Barden's Buffer - Sailing off to C.
William Barden, Jr.

March 1987 page 196
KISSable OS9 - Bootstrapping many systems.
Dale L. Puckett

March 1987 page 194
OS9 Level II - OS9 programming. Finding your way in the new system.
Peter Dibble

April 1987 page 197
KISSable OS9 - Back to the beginning.
Dale L. Puckett

April 1987 page 192
Memory Management - Understanding OS9's memory system.
Peter Dibble

May 1987 page 196
KISSable OS9 - Setting the stage for OS9 Level II.
Dale L. Puckett

May 1987 page 194
Pause for Thought - Avoid mistake lockups with a handy pause command.
Paul Ladoucer

May 1987 page 191
The Advantage of Processes - 'Slicing' programs for greater memory.
Peter Dibble

June 1987 page 148
Fish or Pheish? - Education. Basic09 helps with phoneme recognition.
Del Turner

June 1987 page 162
KISSable OS9 - Shooting for a standard.
Dale L. Puckett

June 1987 page 154
Exploring Level II - A looke at new features from Basic09.
Rick Adams

July 1987 page 100
A Computer's Ancient Native Language - Tutorial. A look at some profound magic for the CoCo.
Peter Dibble

July 1987 page 163
Bits and Bytes of Basic - Basic09 and Level II: Focusing on modules.
Richard White

July 1987 page 167
KISSable OS9 - An OS9 convert speaks out.
Dale L. Puckett

August 1987 page 157
KISSable OS9 - Controller attacks halt line problem.
Dale L. Puckett

August 1987 page 163
The Problem with Basic09 - OS9 Memory. Improving the Editor procedure.
Peter Dibble

September 1987 page 160
KISSable OS9 - Primitive drawing tools.
Dale L. Puckett

September 1987 page 170
Basic09 isn' fast enough - OS9 Programming. Assembly language can be fun.
Peter Dibble

October 1987 page 176
KISSable OS9 - Unlock graphics potential.
Dale L. Puckett

October 1987 page 164
OS9 Programming - Using compressed files.
Peter Dibble

November 1987 page 180
KISSable OS9 - The evolution continues.
Dale L. Puckett

December 1987 page 180
KISSable OS9 - Putting data structures on the drawing board.
Dale L. Puckett

December 1987 page 168
OS9 Programming - Saving and restoring graphics screens.
Peter Dibble

CoCo-Link

CoCo-Link is an excellent magazine to help you with the RSDOS side of the Colour Computer. It is a bi-monthly magazine published by Mr. Robbie Dalzell. Send your subscriptions to:

CoCo-Link
31 Nedlands Crescent
Pt. Noarlunga Sth.
South Australia
Phone: (08) 3861647

Things we'd like to have for Christmas

In recent times, we have talked about "wish-lists" for our favourite computer. Here, we present one such list from Tim Kientzle, one of the more erudite of the contributors to the Internet OS9 List.

I have a few ideas for programs I'd like to see written. Not sure how many of these, if any, I might write myself, but all seem like useful things. They aren't necessarily flashy programs with sophisticated graphics and sound, etc, but they're solid useful programs that would help make life a lot more convenient.

- Interactive Disassembler. A friend of mine wrote a disassembler for his Atari ST that had some really good ideas. It displayed a disassembly on the screen, and you could scroll up and down through it. You could press a key on some line, and everything from there down would be displayed in a different format, such as hex, ASCII, etc. By going through and interactively inserting these markers, you could fairly quickly get a fairly good disassembly of an area. Add in the ability to name labels interactively, and to attach comments to any line, and you'd have one amazingly convenient disassembler.

- Termcap/Curses library. Some are floating around PD, but a "good" library would be very useful. Adding support for things like overlays, menus, etc, on any terminal would make lots of programs a lot easier.

- File picker, editor widgets. A "good" file picker dialogue takes a long time to do well. Ditto for a small editor (say, up to 32k max). Bundled into a library (maybe even with the above Termcap/Curses?), it would be extremely useful.

- Memory allocation library. A handle-based memory manager which can do heap compaction, and can later be extended to support virtual memory concepts would be a great project for some of you CS types. <grin> Basic handle-based memory management is easy to do, but there are tricks when you consider locking blocks of memory etc. malloc() just isn't enough somedays.

- Binary Editor. A good binary file editor would show your file in hex and/or ASCII, or even a combination of several. Should allow insertions and/or deletions, even within a module (automatically updating the module size and such), as well as the standard operations.

- Disk Editor. A "good" disk editor would do more than just display sectors. It should let you walk through the directory heirarchy, let you examine and modify file descriptor sectors (not as a hex dump, either), LSN0, mark sectors as allocated/unallocated in the bitmap, move, rename, delete files, etc.

- Bootfile editor. Should allow convenient mouse-driven insertion and deletion of modules and devices. (i.e. I insert a device, the editor adds the descriptor and the driver and manager, if necessary) Also should allow editing of xmode and tmode settings in descriptors. May or may not include ability to do binary editing of individual modules.

- Spreadsheet. Lotus and the like are messy. Something easy to use, easy to program, with some nice user-oriented features.

- Graphing, data analysis. Someone using Lotus complained to me the other day that there should be a way to grab a point on the graph and have the number in the spreadsheet change to match. This same person was trying to do some curve-fitting as well, which Lotus does not have any convenient facilities for. A nice program which can read in tables of data and display graphs and/or lists of numbers, which are easily changed, graphed in different fashions, etc, could be quite useful some days. If it could also graph equations, etc, that could be even better.

- Mail Agents. I've seen several different programs that attempt to do things like log on to CIS, download all your mail, and log off immediately, leaving the mail in a file where you can read it at your leisure. Ideally, this should of course feed into the existing mail system. i.e. I should be able to send mail on my computer to "72276,1135@compuserve", and have it be spooled to a special directory. Once a week or so, some program is run by "cron" which automatically logs into CIS, mails all my messages, downloads messages to me,

etc. The net effect is that my mail system on my home computer gateways onto CIS, Delphi, UUCP, the Unix system at school etc, etc. Add in full support for the CIS, Delphi forums, and you've got some customers. This would require a lot of work, some sort of sophisticated scripting language, etc, etc.

- "Smart" mail filters. Especially on those days I log in to find 80 messages from the mailing list, along with 10 other messages, I really wish this Unix system supported several mailboxes for each user, with some program which could automatically do some filtering for me, putting all mailing list messages into an inbox called "CoCo", etc, etc.

- Mail reader. A nice, mouse-driven mail reader would be so convenient, especially with the above. Two windows: one showing a list of messages, another showing the messages. Click on a message to read it, buttons/menus to reply to the message, move it to a mail folder, etc. If "news" were available through the same program, it would be even better: I see three types of messages: private person-to-person messages (mail), forum messages available to a narrow group of subscribers (i.e. I don't want anybody on my machine to be able to respond to Delphi Forum messages through my account), and public-distribution (news). Having all three available through one program interface would be quite convenient.

As new machines come out, and more folks have experience with CoCo windows, the need for interesting ideas grows. What programs do you think would make your life easier?

- Tim Keintzle

oooooooooo0000000000ooooooooo

WPShel Product Description

The Australian OS9 User Group does not normally publish advertising material, and this article seems remarkably like just that! However, in the interests of making relevant information available to our readers, here is a product description of WPShel from ColorSystems, in the USA.

WPShel is a Graphics Shell for OS9 Level 2 (OS9) and the Color Computer 3 (CoCo3). It is designed similarly to the Graphics Shell most of you are used to looking at, Multi-Vue's GShell, in that the Main Screen has a "Menu Bar" across the top of the window. Access to all Menu Bar functions is accomplished by pointing the mouse cursor to the Menu Bar "Menu Name" and clicking the left mouse button. There are four Menu Names on the Menu Bar, plus the Tandy Menu Icon. The "Exit Program" icon, a la GShell, is also present.

When you click on a Menu Name, a "pull down" menu is displayed under the Menu Name. To select one of the pull down "Menu Items", you pull down the mouse cursor to the item you want and click the left mouse button. Various actions then occur, based on the specific Menu Item you selected. Overlay windows are used frequently, with input normally either via keyboard entry or mouse click. Most of the more commonly used functions are also accessible via an ALT-key command sequence.

WPShel got it's name because all Menu Names and their associated Menu Items are all Word Processing oriented. There are four Menu Names, "Document", "Print", "Speller", and "Utility". Virtually all basic Shell functions are available via the Menu Items, and just in case there is an OS9 command you need to execute which is not a Menu Item, there is a facility to fork the standard Shell in either another window or in an overlay window.

Since WPShel is a Graphical User Interface (GUI), it will be cumbersome for experianced OS9 Users who prefer to access OS9 at the Command Line level. I originally designed WPShel to be primarily useful for novice OS9 Users who wish to use their system for Word Processing, and for any OS9 user who has the need to set up a "Turnkey" Word Processing system for use by people who can't even spell "OS9". But since the inspriation for WPShel was the highly successful DECB based word processor, TW-128, former TW-128 users may find WPShel enjoyable to use. In fact, early reviewers of WPShel have even referred to it as an "OS9 clone of TW-128".

One VERY important thing to realize about WPShel is that it is a SHELL. It is not an all encompassing Word Processor. The major functions such as the Editor and the Text Formatter are not performed directly by WPShel. They are handed over to an external module in a "forked" process. Big reasons why I designed WPShel in this manner is that

1) Most people are already familiar with an editor and HATE to use any other editor. WPShel allows you to use whichever editor you already have and like to use. It was tested with VED, ED, SLED, EDIT, T/S Edit and DYNASTAR, with VED being the author's editor of choice since it will run in any window type and almost any window size.

2) There are already several excellant editors and text formatters available for OS9, many Public Domain or Shareware.

3) OS9 is structured such that one program is limited to a 64K address space, but the concept of forking a child process to perform work is simple and easy to do.

WPShel is a strong competitor to DynaStar and an even stronger competitor to T/S Word. The only advantage DynaStar has over WPShel is that it has its own builtin editor. It also comes with an external text formatter, df, which will work as WPShel's text formatter, by the way. WPShel's advantages over DynaStar are that it has both a mouse and keyboard interface (DynaStar has no mouse interface) and WPShel includes other user customizable functions like a text file previewer and various spelling checker functions. DynaStar can, in fact, be used as WPShel's editor, but in my opinion, that would be overkill.

WPShel runs rings around T/S Word. WPShel much easier to set up and get going with than T/S Word is. T/S Edit works ok as WPShel's editor (though not as well as some others, VED in particular). T/S Word's text formatter TSFMT doesn't work as a pull down menu option, but can be used in a Shell overlay window. T/S Spell, on the other hand works very well with WPShel, in fact, WPShel's spelling checker functions were explicitly designed to work exceptionally well with T/S Spell.

I have invested a great deal of time and effort into the design, development, and testing of WPShel, and for this reason, I have decided to release it only as a commercial product. The price is inexpensive, though, only $22.

To use WPShel to it's full extent, you will need an editor, a text formatter, a text file previewer utility, and a Spelling Checker. In an appendix in the documentation I mention that I have several Public Domain and Shareware editors, formatters, and text file preview utilities which I can send to you if you ask me for them. For legal reasons I cannot include them with WPShel. I also reference several commercial editors, formatters, and spelling checkers along with current prices and availability, which I recommend.

Also, since WPShel uses the Multi-Vue Menu Bar interface and auto-follow mouse, the WindInt module from your Multi-Vue disk is required in your bootlist. While WPShel can be run as a GShell application (an AIF an Icon are included for that purpose), it doesn't have to be. It will run perfectly fine in any window type from a Shell command. It also runs fine as an "autoex" system function, and included on the WPShel disk is a bootlist for creating a "turnkey" Word Processing Boot Disk, and the documentation contains step by step instructions on how to create this disk.

Also, since overlay windows are used heavily, the "fast gfx" patches to GRFDRV by Kevin Darling and Kent Myers makes the flow of execution much faster and smoother.

WPShel will NOT run on a 128K CoCo3. Since the next increment of memory available for the CoCo3 is the 256K "Quarter-Meg" upgrade from Burke&Burke, the minimum amount of RAM required for WPShel is 256K.

WPShel is ready for delivery now. So, if after reading this, you are still interested in WPShel, ie, want to purchase a copy, send me a check or money order for $22 (I pay all shipping and handling costs) to me. Or if you simply want a catalog of my current offerings of OS-9 Level 2 and OSK-MM/1 software write to me at:

ColorSystems
P.O. Box 540
Castle Hayne, NC 28429
USA

(All software available on 5.25" SSDD, or 3.5" DSDD disk.)

Zack Sessions.

oooooooooo0000000000ooooooooooo

File Size?

As gordon mentioned in his editorial, someone asked how you find the size of a file. Of course, everyone answers, use DIR E (or dir -e if you use the newer dir command), but when I looked at the question again, I was struck by the possibility that maybe the question should have been read as 'how do you find the size of ONE file?' Well, the dir command gives you a whole directory full, of course, and in HEXADECIMAL to boot, which may be more than you bargained for. If the file is a (single) module, you could use IDENT. Again, more information than you really needed. So, OK then, how DO you find it? Well, of course, use Bob Devries' magical FileSize programme. What's that? Never heard of it? Well that's not too surprising really, seeing as I just wrote it.

How does it work? Well, it is not too complicated. First it tries to open the file you asked for as a FILE, if that fails, it tries to open it as a DIRECTORY (you never know...). Then it uses an (undocumented) OS9 I$GETSTT system call SS_FD, which reads the file descriptor, and stores it into a memory area (structure). When that is done, it reads the necessary bytes, four of them actually, to find the length of the file. The names in the structure 'fildes' which I used are defined in the

header file 'direct.h', which is on the C compiler disk in the DEFS directory. Then it uses the C printf call to print both the filename and the length, in DECIMAL.

Note the second version of the readfd() function. This is provided for two reasons, 1. to show an alternate method of doing the same thing, and 2. to provide portability to other OS9 systems (e.g. OSK)

See, simple isn't it. Here's the C source code.

Regards, Bob Devries

```
/* FileSize */
/* Get the size of a file */
/* by Bob Devries 16 Nov 91 */

#include <stdio.h>
#include <direct.h>
#include <os9.h>

FILE *fp, *fopen();     /* declare file pointer etc */
struct fildes desc;     /* declare file descriptor structure */

main(argc,argv)
int argc;
char *argv[];
{
    pflinit();          /* we will be printing longs */

    if (argc != 2) {    /* not two command line args? */
        puts("Usage: filesize <file>");    /* tell user he erred */
        exit(0);        /* and exit nicely */
    }

    if ((fp = fopen(argv[1],"r")) == NULL) {          /* open read */
        if ((fp = fopen(argv[1],"d")) == NULL) {      /* or open read_dir */
            printf("Can't open %s.\n",argv[1]);       /* no? so say so */
            exit(errno);                              /* quit with error */
        }
    }

    readfd();           /* go away and read the file desc into structure */
                        /* and report size. That's what we're here for   */

    printf("File %s is %ld bytes long.\n",argv[1],desc.fd_fsize);
}

readfd()
{
    struct registers regs;        /* set up temp struct to contain CPU regs */

    regs.rg_a = fileno(fp);       /* A = path number */
    regs.rg_b = SS_FD;            /* B = value for SS_FD system call */
    regs.rg_x = &desc;            /* X = pointer to structure */
    regs.rg_y = sizeof(desc);     /* Y = length of struct */
    regs.rg_u = 0;
    _os9(I_GETSTT,&regs);         /* use I$GETSTT to do it */
}
```

alternate readfd() function
if you have Carl Kreider's C Library

```
readfd()
{

    getstat(SS_FD,fileno(fp),&desc,sizeof(desc));

}
```

oooooooooo0000000000oooooooooo

Seasons Greetings
from
National OS9 Users Group